排序(以下默认从小到大排序)

1. 冒泡排序
2. 算法简介
3. 它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这是一种稳定的排序算法，时间复杂度为(n2)，效率非常低，做题时不推荐使用。
4. 算法描述

1、比较相邻的元素。如果第一个比第二个大，就交换他们两个。

2、对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。

3、针对所有的元素重复以上的步骤，除了最后一个。

4、持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

3) 算法演示

初始状态： 49 38 65 97 76 13 27 49

第一趟排序：38 49 65 76 13 27 49 97

第二趟排序：38 49 65 13 27 49 76 97

第三趟排序：38 49 13 27 49 65 76 97

第四趟排序：38 13 27 49 49 65 76 97

第五趟排序：13 27 38 49 49 65 76 97

第六趟排序：13 27 38 49 49 65 76 97

第七趟排序：13 27 38 49 49 65 76 97

1. 代码描述

void bubble\_sort(int n) {

int flag = 1;

while(n > 1) {//对于n个数只需要排序n-1次

flag = 1;

for(int i = 0; i < n-1; i++) {

if(a[i] > a[i+1]) {//相邻两个数进行比较

int temp = a[i];

a[i] = a[i+1];

a[i+1] = temp;

flag = 0;

}

}

if(flag) break;

n--;

}

}

1. 插入排序

## 算法简介

插入排序（Insertion Sort）的算法是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。这是一种稳定的排序算法，时间复杂度为(n2)。

## 算法描述

    一般来说，插入排序都采用in-place在数组上实现。具体算法描述如下：

    1、从第一个元素开始，该元素可以认为已经被排序

    2、取出下一个元素，在已经排序的元素序列中从后向前扫描

    3、如果该元素（已排序）大于新元素，将该元素移到下一位置

    4、重复步骤3，直到找到已排序的元素小于或者等于新元素的位置

    5、将新元素插入到该位置后

   6、重复步骤2~5

3) 算法演示

初始状态： 49 38 65 97 76 13 (49默认有序)

第一趟排序：38 49 65 97 76 13 (38插入已排序序列中)

第二趟排序：38 49 65 97 76 13 (65已经有序)

第三趟排序：38 49 65 97 76 23 (97已经有序)

第四趟排序：38 49 65 76 97 13 (76插入已排序序列中)

第五趟排序：13 38 49 65 76 97 (13插入已排序序列中)

1. 代码描述

void insert\_sort(int n) {//数组的值从下标1开始

， int i, j;

for(int i = 2; i <= n; i++) {

int j = i-1;

a[0] = a[i];//用a[0]记录a[i]的值

while(a[0] < a[j]) {//已排序数据从后往前，大于a[i]的就往后移

a[j+1] = a[j];

j--;

}

a[j+1] = a[0];//把a[i]放到合适的位置

}

}

1. 选择排序

## 算法简介与描述

选择排序是一种简单直观的排序算法。它的工作原理如下。首先在未排序序列中找到最小元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。这是一种不稳定的排序算法，时间复杂度为(n2)。

1. 算法演示

初始状态： 3 6 4 2 11 10 5

第一趟排序：2 6 4 3 11 10 5 (2在未排序序列中最小，与3交换)

第二趟排序：2 3 4 6 11 10 5 (3在未排序序列中最小，与6交换)

第三趟排序：2 3 4 6 11 10 5 (4在未排序序列中最小，位置不变)

第四趟排序：2 3 4 5 11 10 6 (5在未排序序列中最小，与6交换)

第五趟排序：2 3 4 5 6 10 11 (6在未排序序列中最小，与11交换)

第六趟排序：2 3 4 5 6 10 11 (10在未排序序列中最小，位置不变)

1. 代码描述

void select\_sort(int n) {

for(int i = 0; i < n-1; i++) {

int t = i;

for(int j = i+1; j < n; j++) {//找到未排序序列中最小值的下标

if(a[j] < a[t]) t = j;

}

int temp = a[i];

a[i] = a[t];

a[t] = temp;

}

}

1. 分治法

在计算机科学中，分治法是一种很重要的算法。字面上的解释是“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。这个技巧是很多高效算法的基础，如[排序算法](https://baike.baidu.com/item/%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95" \t "https://baike.baidu.com/item/%E5%88%86%E6%B2%BB%E6%B3%95/_blank)([归并排序](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E5%88%86%E6%B2%BB%E6%B3%95/_blank)，[快速排序](https://baike.baidu.com/item/%E5%BD%92%E5%B9%B6%E6%8E%92%E5%BA%8F" \t "https://baike.baidu.com/item/%E5%88%86%E6%B2%BB%E6%B3%95/_blank))。

使用分治法解决问题一般分为以下三个步骤：

划分问题：把问题的实例划分成子问题。

递归求解：递归解决子问题。

合并问题：合并子问题的解得到原问题的解

1. 归并排序
2. 算法简介

  归并排序是建立在归并操作上的一种有效的排序算法。该算法是采用分治法的一个非常典型的应用。 将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。这是一种稳定的排序算法，时间复杂度为(nlogn)。

1. 算法描述

安照分治三步法，可以把归并排序做以下描述

划分问题：把序列分成元素个数尽量相等的两半。

递归求解：把两半元素分别排序。

合并问题：把两个有序表合并成一个。

1. 算法演示

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15 | 12 | 14 | 13 | 11 | 16 |

|  |  |  |
| --- | --- | --- |
| 15 | 12 | 14 |

|  |  |  |
| --- | --- | --- |
| 13 | 11 | 16 |

|  |  |
| --- | --- |
| 15 | 12 |

|  |
| --- |
| 15 |

|  |
| --- |
| 12 |

|  |
| --- |
| 14 |

|  |  |
| --- | --- |
| 13 | 11 |

|  |
| --- |
| 16 |

分割

|  |
| --- |
| 14 |

|  |
| --- |
| 13 |

|  |
| --- |
| 16 |

|  |
| --- |
| 11 |

|  |  |
| --- | --- |
| 12 | 15 |

|  |  |
| --- | --- |
| 11 | 13 |

|  |
| --- |
| 14 |

|  |
| --- |
| 16 |

|  |  |  |
| --- | --- | --- |
| 12 | 14 | 15 |

|  |  |  |
| --- | --- | --- |
| 11 | 13 | 16 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 11 | 12 | 13 | 14 | 15 | 16 |

合并

1. 代码描述

void merge(int x, int mid, int y) {//合并

int p = x, q = mid, i = x;

while(p < mid && q < y) {

if(a[p] <= a[q]) t[i++] = a[p++];

else t[i++] = a[q++];

}

while(p < mid) t[i++] = a[p++];

while(q < y) t[i++] = a[q++];

for(int i = x; i < y; i++) a[i] = t[i];

}

void merge\_sort(int x, int y) {

if(y > x+1) {

int mid = (x+y)/2;//划分

merge\_sort(x, mid);//递归求解

merge\_sort(mid, y);//递归求解

merge(x, mid, y);//合并有序表

}

}

1. 快速排序
2. 算法简介

快速排序是最快的通用内部排序，它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以[递归](https://baike.baidu.com/item/%E9%80%92%E5%BD%92" \t "https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95/_blank)进行，以此达到整个数据变成有序[序列](https://baike.baidu.com/item/%E5%BA%8F%E5%88%97/1302588" \t "https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95/_blank)。这是一种不稳定的排序算法，时间复杂度为(nlogn)。

1. 算法描述

安装分治三步法，可以把归并排序做以下描述

划分问题：把数组的各个元素重排后分成左右两部分，使得左边任意元素都小于或等于右边的任意元素。

递归求解：把左右两部分分别排序。

合并问题：不要合并，因为此时数组已经完全有序。

1. 算法演示

快速排序有很多种划分方法，这里举出其中一种，以序列的最后一个数为界限，设为m，小于等于m的放左边，否则放右边。开始时设i为这个序列的第一个下标0，从左往右遍历这个序列，如果当前位置的值小于等于m就与a[i]交换，并且i++，否则不做任何操作，遍历完后，最后再把m和a[i]交换，这样就达到了划分的目的。

初始状态：67 23 89 35 28 90 10 24

第一次划分：

开始时i = 0；用j 遍历数组;

j = 0：67 23 89 35 18 90 10 24 (67大于24不做任何操作)

j = 1: 23 67 89 35 18 90 10 24 (23小于24，与a[i](67)交换，i++)

j= 2：23 67 89 35 18 90 10 24 (89大于24不做任何操作)

j = 3：23 67 89 35 18 90 10 24 (35大于24不做任何操作)

j = 4：23 18 89 35 67 90 10 24 (18小于24，与a[i](67)交换，i++)

j = 5: 23 18 89 35 67 90 10 24 (90大于24不做任何操作)

j = 6: 23 18 10 35 67 90 89 24 (10小于24，与a[i](89)交换，i++)

最后把24与a[i](35)交换，第一次划分完成

即：23 18 10 24 67 90 89 35

再分别对23 18 10和24 67 90 89 35进行以上操作

1. 代码描述：

int partition(int x, int y) {

int m = a[y-1];//把序列最后一个数赋值给m

int i = x;

for(int j = x; j < y - 1; j++) {

if(a[j] <= m) {

int t = a[i]; a[i] = a[j]; a[j] = t; //交换a[i]和a[j]

i++;

}

}

int t = a[y-1]; a[y-1] = a[i]; a[i] = t;//最后把m和a[i]交换

return i;

}

void quick\_sort(int x, int y) {

if(y > x+1) {

int mid = partition(x, y);//划分

quick\_sort(x, mid);//递归求解

quick\_sort(mid, y);//递归求解

}

}

练习：

hdu 1872 稳定排序

hdu 1106 排序

hdu 3789 奥运排序问题

hdu 1425 sort

hdu 5775 Bubble Sort

hdu 4911 Inversion